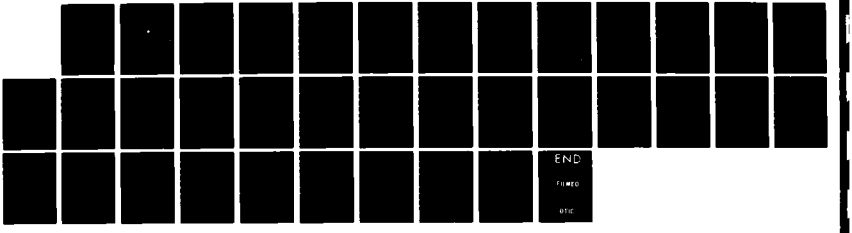
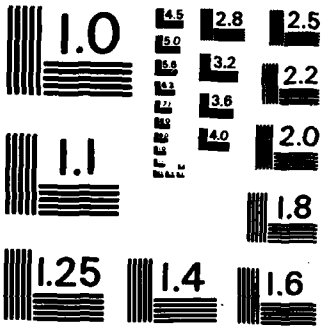
AD-A161 196 MINIMIZING FILL-IN FOR GAUSSIAN ELIMINATION OF SYMMETRIC SYSTEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA Y H SEO SEP 85 1/1

UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

AD-A161 196

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
ELECTE
NOV 19 1985
S D
A

THESIS

MINIMIZING FILL-IN FOR GAUSSIAN ELIMINATION
OF
SYMMETRIC SYSTEMS

by

Young Ho Seo

September 1985

Thesis Advisor: R. K. Wood

Approved for public release; distribution unlimited

DTIC FILE COPY

85 11 19 169

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Minimizing Fill-In for Gaussian Elimination of Symmetric Systems		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1985 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Young Ho Seo		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		12. REPORT DATE September 1985 13. NUMBER OF PAGES 34
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Gaussian Elimination, Choleski Factorization, Fill-in, Minimum Degree Ordering

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The new projective linear programming algorithm by Karmarkar requires solution of symmetric, positive definite systems of equations. One key to solving these systems efficiently is minimizing the fill-in; i.e., the number of new nonzero elements in the reduced matrix, in the various forms of Gaussian elimination used to solve these systems. Fill-in is affected only by symmetric reordering of rows and columns of the system of —

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 Contd.

equations. Various heuristic ordering algorithms are tested and compared with the heuristic minimum degree algorithm of George and Liu. Computational results are reported for sixteen large-scale real-world and artificial problems. The minimum degree algorithm of George and Liu is the most effective of six tested methods.

Approved for public release; distribution is unlimited.

Minimizing Fill-In for Gaussian Elimination of Symmetric Systems

by

Young Ho Seo
Major, Republic of Korea Army
B.S., Korea Military Academy, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1985

Author: Seo, Young Ho
Young Ho Seo

Approved by: K. T. Wood
Kevin Wood, Thesis Advisor

Gerald G. Brown, Second Reader

Alan R. Washburn, Chairman,
Department of Operations Research

K.T. Marshall
Kneale T. Marshall,
Dean of Information and Policy Sciences

Accesion For
NTIS CRA&
A1

ABSTRACT

The new projective linear programming algorithm by Karmarkar requires solution of symmetric, positive definite systems of equations. One key to solving these systems efficiently is minimizing the fill-in, i.e., the number of new nonzero elements in the reduced matrix, in the various forms of Gaussian elimination used to solve these systems. Fill-in is affected only by symmetric reordering of rows and columns of the system of equations. Various heuristic ordering algorithms are tested and compared with the heuristic minimum degree algorithm of George and Liu. Computational results are reported for sixteen large-scale real-world and artificial problems. The minimum degree algorithm of George and Liu is the most effective of six tested methods.

TABLE OF CONTENTS

I. INTRODUCTION 6

A. INTRODUCTION 6

B. PROJECTIVE LP ALGORITHM 6

C. DIRECT SOLUTION METHODS 7

1. Basic Gaussian Elimination 8

2. Cholesky Factorization 9

3. QR Factorization 11

D. GRAPH THEORETIC MODEL 12

E. CURRENT METHODS 15

F. TEST PROBLEMS 16

II. THE MINIMUM DEGREE ALGORITHM 18

A. INTRODUCTION 18

B. BASIC ALGORITHM 18

C. ENHANCED ALGORITHM 19

D. IMPLEMENTATION 20

III. THE MINIMUM FILL-IN ALGORITHM AND VARIANTS 22

A. INTRODUCTION 22

B. THE MINIMUM FILL-IN ALGORITHM 22

C. IMPLEMENTATION OF THE MINIMUM FILL-IN ALGORITHM 24

D. MIN DEGREE / MIN FILL-IN ALGORITHM 25

E. OTHER VARIANTS 26

IV. COMPUTATIONAL EXPERIENCE AND DIFFICULTIES 27

V. CONCLUSIONS AND RECOMMENDATIONS 31

LIST OF REFERENCES 32

INITIAL DISTRIBUTION LIST 33

LIST OF TABLES

I	PROBLEM DIMENSIONS	17
II	RESULTS OF ORDERING ALGORITHMS	28
III	FILL-IN RESULTS	29

I. INTRODUCTION

A. INTRODUCTION

The new projective linear programming algorithm by Karmarkar [Ref. 1] requires solution of symmetric, positive definite systems of equations. One of the keys to solving these systems efficiently is controlling the fill-in, i.e., the number of new nonzero elements in the reduced matrix, in the various forms of Gaussian elimination used to solve these systems. Under the assumption of noncancellation of terms, fill-in created by direct solution techniques is affected only by symmetric reordering of the rows and columns of the system to be solved.

The problem of minimizing fill-in by symmetric reordering is known to be NP-complete [Ref. 9] so, in this thesis, we examine a number of heuristic reordering algorithms for this problem.

Several different algorithms for reducing fill-in in Gaussian elimination are compared for sixteen large-scale matrices derived from linear and integer programming problems.

B. PROJECTIVE LP ALGORITHM

Consider the linear programming problem

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Reproduced from best available copy.

where A is an m-by-n matrix, c is an n-element column vector and b is an m-element column vector. Karmarkar's algorithm requires the solution of $BB^Tz = Bc$, where

$$B = \begin{vmatrix} AE \\ \underline{1}^T \end{vmatrix}$$, E is an n-by-n diagonal matrix, and $\underline{1}$ is

a n-element column vector of 1s. Thus,

$$BB^T = \begin{vmatrix} AE^2A^T & A\underline{E1} \\ \underline{1}^TEA^T & n \end{vmatrix}.$$

We will not be concerned with the bottom row and rightmost column; they will typically be 100% dense when using an artificial starting solution and will be placed last in any ordering. Under the assumption of noncancellation of terms, the nonzeros created in AA^T will be the same as in AE^2A^T, both when creating these matrices and when performing the elimination. Thus, we may restrict study to solving systems of the form $AA^Tz = Ac = b'$.

C. DIRECT SOLUTION METHODS

Symmetry and positive definiteness are important in computation with the matrices. The matrix D is symmetric when $D^T = D$. Otherwise, D is asymmetric. A symmetric matrix D is said to be positive definite if $z^TDz > 0$ for all nonzero vectors z.

Three basic methods are available for direct solution of positive definite systems of the form $AA^Tz = Dz = b'$. These methods are basic Gaussian elimination [Ref. 6], Cholesky factorization [Ref. 4] and QR factorization [Ref. 7]. All three methods end up creating an upper or lower triangular matrix which is used with b', or a modified version of b', to solve for z by backward and/or forward substitution.

One key to success of these methods is keeping the triangular matrix as sparse as possible by symmetrically reordering the rows and columns of D. This keeps computation time down and round-off errors to a minimum.

1. Basic Gaussian Elimination

The Gaussian elimination method we will consider for the solution of a set of equations is just an generalization of the familiar method of eliminating one unknown by substitution between a pair of simultaneous equations and is the basic pattern of a large number of solution methods that can be classed as direct methods.

Let us consider the system $Dz = b'$, represented by the augmented matrix $[D,b']$, where D is an m-by-m positive definite symmetric matrix and b' is an m-vector. Gaussian elimination by columns consists of m steps. The purpose of the kth step is to eliminate all the nonzero elements of the augmented matrix which lie on column k below the diagonal of D. At the kth step, $d^{(k)}_{kk}$ which is the diagonal of the active submatrix (formed by rows k to m and column k to m of $D^{(k)}$) is selected to be the pivot. Row k of the augmented matrix is normalized and the elements in column k below the diagonal of $D^{(k)}$ are eliminated by subtraction of multiples of the normalized row k from all those rows which have a nonzero in column k below the diagonal. The augmented system $[D^{(k)},b'^{(k)}]$ is obtained where $D^{(k)}$ has zeros in its first k columns below the diagonal and 1s on the first k positions of the diagonal.

The process is continued until the upper triangular system $[D^{(m)},b'^{(m)}]$ is obtained and z can be solved for by backward substitution.

2. Cholesky Factorization

There exists a unique lower triangular matrix L such that $LL^T = D$. Cholesky factorization creates L from D directly without consideration of b'. Then, $LL^Tz = Dz = b'$ is solved for by:

(1) Solve $Ly = b'$ for y.

(2) Solve $L^Tz = y$ for z.

Suppose a positive definite symmetric m-by-m matrix D is given. The way in which fill-in occurs when computing L is best illustrated by showing a proof, following George and Liu [Ref. 4], of one of the Cholesky factorization techniques.

Let $D = \begin{vmatrix} d_1 & a_1^T \\ a_1 & K_1 \end{vmatrix}$ where d_1 is a scalar, a_1 is an m-1 vector and K_1 is an (m-1)-by-(m-1) submatrix. Then we can write

$$D = \begin{vmatrix} d_1^{1/2} & 0 \\ a_1/d_1^{1/2} & I_{m-1} \end{vmatrix} \begin{vmatrix} 1 & 0 \\ 0 & H_1 \end{vmatrix} \begin{vmatrix} d_1^{1/2} & a_1^T/d_1^{1/2} \\ 0 & I_{m-1} \end{vmatrix}$$

$$= L_1 \begin{vmatrix} 1 & 0 \\ 0 & H_1 \end{vmatrix} L_1^T$$

$$= L_1 D_1 L_1^T,$$

where $H_1 = K_1 - a_1 a_1^T/d_1$, and I_k denotes an identity

matrix of order k. Then,

$$D_1 = \begin{vmatrix} 1 & 0 \\ 0 & A_1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & d_2 & a_2^T \\ 0 & a_2 & K_2 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 0 & \\ 0 & d_2^{1/2} & \\ 0 & a_2/d_2^{1/2} & I_{m-2} \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & H_2 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & d_2^{1/2} & a_2^T/d_2^{1/2} \\ 0 & & I_{m-2} \end{vmatrix}$$

where $H_2 = K_2 - a_2 a_2^T/d_2$

$$= L_2 D_2 L_2^T,$$

$$\vdots$$

$$D_{m-1} = L_m T_m L_m^T.$$

Here, for $1 \le i < m$, d_i is a positive scalar, a_i is a vector of length m-i, and H_i is an (m-i)-by-(m-i) positive definite symmetric matrix.

After m steps of the algorithm, we have

$$D = L_1 L_2 \cdots L_m L_m^T \cdots L_2^T L_1^T = LL^T,$$

where it can be shown that

$$L = L_1 + L_2 + \cdots + L_m - (m-1)I_m.$$

Thus, the ith column of L is precisely the ith column of L_i.

In this scheme, the columns of L are computed one by one. At the same time, each step involves the modification

of the submatrix K_i by the outer product $a_i a_i^T/d_i$ to give H_i, which is the submatrix remaining to be factored. Thus, the structure of H_i is different from K_i due to $a_i a_i^T/d_i$. $a_i a_i^T/d_i$ creates new fill-ins if $(a_i)_K (a_i)_j \neq 0$ but $(H_i)_{Kj} = 0$ where $(a_i)_K$ is a kth element of vector a_i and $(H_i)_{Kj}$ is the scalar element in row k and column j if H_i.

It can be seen from the above derivation that the matrices $D^{(k)}$ and H_K correspond in their patterns of nonzeros, and if the normalization step of standard Gaussian elimination is replaced by "divide row k by $(d^{(k)}_{KK})^{1/2}$," then the upper triangular matrix produced is actually L^T. Under the assumption of noncancellation of terms, the pattern of nonzeros in L^T and the upper triangular portion of $D^{(m)}$ will be the same.

3. QR Factorization

For completeness let us also consider QR orthogonal factorization. Orthogonal factorization methods compute the triangular Cholesky factor L directly from A rather than AA^T and thus avoid explicit formation of the cross product used in basic Gaussian elimination and Cholesky factorization. In QR factorization, an orthogonal matrix Q of order n is computed which reduces A and c to the form

$$Q^T A^T = \begin{vmatrix} L^T \\ 0 \end{vmatrix} \quad \text{and} \quad Q^T c = \begin{vmatrix} q \\ r \end{vmatrix},$$

where q is a vector of dimension m, and r is a vector of dimension m-n.

Then, $Dz = b'$ is equivalent to

$$AA^T z = Ac,$$

$$\Longrightarrow \quad AQQ^T A^T z = AQQ^T c,$$

$$\Longrightarrow \quad [L \;\; 0] \begin{vmatrix} L^T \\ 0 \end{vmatrix} z = [L \;\; 0] \begin{vmatrix} q \\ r \end{vmatrix},$$

$$\Longrightarrow \quad LL^T z = Lq,$$

$$\Longrightarrow \quad \text{and } L^T z = q.$$

Thus, after computing L^T and q using the orthogonal matrix Q, we need only solve for z by backward substitution.

D. GRAPH THEORETIC MODEL

Consider the two matrices below where x indicates a nonzero. If we perform Gaussian elimination on the two matrices, we get the upper triangular forms shown.

$$\begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & x & x & x & x \\ 2 & x & x & & \\ 3 & x & & x & \\ 4 & x & & & x \end{array} \quad \Longrightarrow \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & x & x & x & x \\ 2 & & x & x & x \\ 3 & & & x & x \\ 4 & & & & x \end{array} \qquad (1)$$

$$\begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & x & & & x \\ 2 & & x & & x \\ 3 & & & x & x \\ 4 & x & x & x & x \end{array} \quad \Longrightarrow \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & x & & & x \\ 2 & & x & & x \\ 3 & & & x & x \\ 4 & & & & x \end{array} \qquad (2)$$

Under assumption of noncancellation of terms, the Cholesky factor of (1) is 100% dense. If the first and last columns of (1) are interchanged, and the first and last rows of (1) are interchanged, then Gaussian elimination will produce no fill-in at all. Below, a graph theoretic model is described

[Ref. 3] which helps clarify the process of creating fill-ins and how this can be modified by reordering.

An undirected graph $G=(V,E)$ consists of a finite set of vertices V together with a set E of edges, where an edge is an unordered pair of vertices (u,v) such that $u \neq v$. An undirected graph may be associated with any symmetric matrix D of order n. The "adjacency graph" of D is a graph $G=(V,E)$ such that $V=\{v_1,\ldots,v_n\}$ and $(v_i,v_j) \in E$ if and only if $d_{ij} \neq 0$ for all $i \neq j$. The adjacency graph $G^{(k)}$ associated with the active submatrix of $D^{(k)}$ is called an "elimination graph." Note that an adjacency graph G associated with D is unaffected by any symmetric reordering of the rows and columns D.

If (u,v) is an edge, u and v in G are "adjacent." For $S \subset V$, the adjacent set of S, denoted by $Adj(S)$, is the set of all vertices not in S which are adjacent to vertices in S. That is,

$$Adj(S) = \{u \in V-S \mid (u,v) \in E \text{ for some } v \in S\}$$

where $V-S$ is the set of all vertices of V which are not in S.

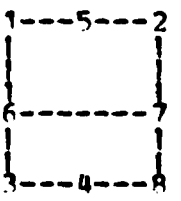
A subgraph $G' = (V',E')$ of $G = (V,E)$ is a graph which consists of some or all vertices of G and some of the edges of G: $V' \subseteq V, E' \subset E$. A subgraph is a clique if every pair of vertices in a subgraph is adjacent.

Let us consider a symmetric sparse matrix and its associated undirected graph. The matrix is of order 8.

$D = D^{(0)} =$

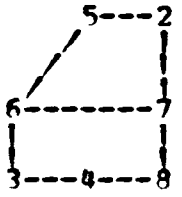
	1	2	3	4	5	6	7	8
1	x				x	x		
2		x			x		x	
3			x	x			x	
4			x	x				x
5	x	x			x			
6	x		x			x	x	
7		x				x	x	x
8				x			x	x

$G^{(0)} =$



If we perform Gaussian elimination at column 1, we obtain $D^{(1)}$.

$$D^{(1)} = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & - & & & & - & & & - \\ 2 & & x & & & & x & & x \\ 3 & & & x & x & & & x & \\ 4 & & & x & x & & & & x \\ 5 & & x & & & x & o & & \\ 6 & & & x & & o & x & x & \\ 7 & & x & & & & x & x & x \\ 8 & & & & x & & & x & x \end{array}$$

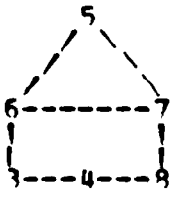


where new fill-ins are noted by circles and - corresponds to a nonzero in the nonactive part of $D^{(k)}$. Fill-in in $D^{(1)}$ corresponds to edges of $G^{(1)}$ which were not in $G^{(0)}$. $G^{(1)}$ is created from $G^{(0)}$ by deleting v_1 and all incident edges and by then connecting all vertices which were adjacent to v_1 and not already connected. This is formalized below.

Let $G^{(k)}$ be an elimination graph. If vertex v_i corresponding to row i and column i is next selected for elimination, then the new fill-in created will be equal to the number of edges in the complement of the graph induced from $G^{(k)}$ by $Adj(v_i)$. A graph $G'(V',E')$ is "induced" from $G(V,E)$ by $V' \subseteq V$ if $E' = E - \{(u,v) \in E$ s.t. $u \notin V'$ or $v \notin V'\}$. $G'' = (V,E'')$ is the "complement" of G if $E'' = \{(u,v)$ s.t. $u, v \in V, u \neq v\} - E$.

The next elimination step is shown below for completeness.

$$D^{(2)} = \begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & - & & & & - & & & - \\ 2 & & - & & & & - & & - \\ 3 & & & x & x & & & x & \\ 4 & & & x & x & & & & x \\ 5 & & & & & x & o & & \\ 6 & & x & & & o & x & x & \\ 7 & & & & & & x & x & x \\ 8 & & & & x & & & x & x \end{array}$$



The notion of "reachable sets" is useful for the study of Gaussian elimination (George and Liu, 1978a). Given a graph and a subset S of vertices, if u and v are two distinct vertices which do not belong to S, we say that v is

reachable from u through S when u and v are connected by a path which is either of length 1 (u and v are adjacent) or is composed entirely of vertices which belong to S. Given S and $u \notin S$, the reachable set Reach(u,S) of u through S is defined to be the set of all vertices which are reachable from u through S. When S is empty, or when u does not belong to Adj(S), then Reach(u,S) = Adj(u).

A tree is an undirected acyclic graph which is connected and has no cycles. A matrix may have a graph which is a tree. If the tree is labelled in increasing order of vertex degree and the matrix is permuted accordingly, then Gaussian elimination with diagonal pivoting can be performed without any fill-in being produced. When the graph of a matrix is not a tree, then it is possible to find a tree partitioning of the graph, i.e., a partitioning such that the "quotient graph" is a tree. Let the set of vertices V of a graph G be partitioned into the disjoint subsets $\eta_1, \eta_2, \ldots, \eta_K$. The quotient graph has the k sets η_i as composite vertices, and its edges are defined by the condition that (η_i, η_j) is an edge if and only if two vertices $u \in \eta_i$ and $v \in \eta_j$ can be found such that u and v are adjacent in G. When the quotient graph is a tree, it is called a "quotient," and the corresponding partitioning a "tree partitioning."

E. CURRENT METHODS

A number of reordering methods are known which dramatically reduce the fill-in resulting from factorization. These include the minimum degree algorithm, various dissection schemes, and various bandwidth or profile ordering schemes.

Band and profile methods regard the fill-in in a global manner, and confine it to certain areas, i.e., the band or profile of the matrix.

The dissection ordering method is a method for systematically partitioning the graph associated with a matrix using separators. In the dissection method, the fill-in caused by Gaussian elimination with diagonal pivoting is confined to nested partitioned areas of the matrix.

It is difficult to compare methods because the efficiency of a method depends on the problem or class of problems for which it is being used. Thus, we will only be concerned with algorithms which do not exploit special structure except implicitly. The method which is currently accepted as best for "unstructured" problems is the "minimum degree algorithm" of George and Liu [Ref. 4]. This thesis tests for efficiency the George and Liu algorithm and several other algorithms including variants of the George and Liu algorithm.

F. TEST PROBLEMS

The algorithms described in this thesis are tested on sixteen linear programming and mixed integer programming problems. These problems consist of fourteen real-world problems (AIR, COAL, CUPS, BUS, FLEC, FOAM, JCAP, NETTING, PASTOR1, PAD, PIES, PILOT, STEEL, TIGER) and two artificial problems (STEINFR1, STEINFR2). The dimensions of these problems are shown in Table I.

All programs are written in FORTRAN, compiled under FORTRAN H-Extended (Opt 2) and run on an IBM 3033AP under VM/CMS. Computation times presented in the following chapters are accurate to the number of decimal places shown.

TABLE I

PROBLEM DIMENSIONS

Problem	Rows	Columns	NZEL [1]	Density	NCE [2]	m*(m-1)/2	Model
AIR	170	3040	6023	1.2%	2.0	14365	Pysical Distribution
BUS	56	530	3365	11.3	6.3	1540	Bus routing (Set Covering)
COAL	170	3753	7506	1.2	2.0	14365	National Energy Planning
CUPS	360	618	1341	0.6	2.2	64620	Production Scheduling
ELEC	784	2800	8462	0.4	3.0	306936	Energy Production & Consumption
FCAM	999	4020	13083	0.3	3.3	498501	Production Scheduling
JCAP	2486	3849	9510	0.1	2.5	3088855	Prod. & Shipment Scheduling
NETTING	89	190	388	2.3	2.0	3916	International Currency Exchange
PASTOR1	657	2074	15894	0.8	5.2	215496	Combinatorics Problem
PAD	694	3297	15541	0.1	4.7	240471	Energy Alloc., Dist. & Consump.
PIES	662	3011	13376	0.7	4.4	218791	Energy Prdn. & Cspn.
PILOT	974	2172	12927	0.6	6.0	473851	Energy Development Planning
STEEL	831	1276	9808	0.9	7.7	344865	Econometric Production & Dist.
STEINER1	117	27	352	11.1	13.0	6786	Artificial Data
STEINER2	330	45	991	6.7	22.0	54285	Artificial Data
TIGER	160	636	4134	4.1	6.7	12720	Set Covering

1 NZEL is the total number of nonzeros in the m by n constraint matrix.
2 NCE is the average number of nonzeros in each column.

II. THE MINIMUM DEGREE ALGORITHM

A. INTRODUCTION

In this chapter we discuss the minimum degree algorithm, with details of the implementation by George and Liu [Ref. 8]. This algorithm is the most commonly used ordering algorithm which does not explicitly exploit special matrix structures. It is used here as a standard for comparison against the algorithms given in chapter III.

The main idea of the minimum degree algorithm is that when we perform Gaussian elimination we choose the vertex of minimum degree from the elimination graph at every step. This myopically minimizes the total fill-in versus the new fill-in. If there are several minimum degree vertices at a particular step, ties are broken arbitrarily.

B. BASIC ALGORITHM

Algorithm : The Basic Minimum Degree Algorithm.

Input : Elimination graph $G^{(0)}$ associated with a positive definite symmetric matrix.

Output : A permutation vector PERM( ) for the vertices of $G^{(0)}$

Step 1. (Initialization)
Set i = 1.

Step 2. (Select minimum degree vertex)
In the elimination graph $G^{(i-1)} = (V_{i-1}, E_{i-1})$, select a vertex $v_k \in V_{i-1}$ which is of minimum degree.

Step 3. (Transform the graph)

Form the new elimination graph $G^{(i)} = (V_i, E_i)$

by eliminating the vertex v_k from $G^{(i-1)}$,

where all vertices adjacent to v_k in $G^{(i-1)}$

are connected to form a clique.

Set PERM(i) = k.

Step 4. (Loop or stop)
Set i = i + 1.
If i > |V|, go to step 5.
Otherwise, go to step 2.

Step 5. (Output)
For j = 1 to |V|, print PERM(j).

End of The Basic Minimum Degree Algorithm.

C. ENHANCED ALGORITHM

We can enhance the minimum degree algorithm by using the concept of "indistinguishable" vertices by George and Liu(1981). Two unlabelled vertices u, $v \in V-S$ are said to be indistinguishable if $Reach(u,S) \cup \{u\} = Reach(v,S) \cup \{v\}$. At any step k, if we find indistinguishable vertices they can be labelled next without any minimum degree search.

Algorithm : The Enhanced Minimum Degree Algorithm.

Input : Elimination graph G(0) associated with a positive definite symmetric matrix.

Output : A permutation vector PERM( ) for the vertices of G(0)

Step 1. (Initialization)

Set i = 1.

Step 2. (Select minimum degree vertex)

In the elimination graph $G^{(i-1)} = (V_{i-1}, E_{i-1})$,

select a vertex $v_k \in V_{i-1}$ which is of minimum

degree.

Step 3. (Find the indistinguishable vertices)

Find the vertices which are indistinguishable

from v. Let Y be the set of v and all vertices

indistinguishable from v.

Step 4. (Elimination)

For each vertex u $\in$ Y: Eliminate u in $G^{(i-1)}$

to obtain $G^{(i)}$, set PERM(i) = u, and

set i = i + 1.

Step 5. (Loop or stop)

If i > |V|, go to step 6.

Otherwise go to step 2.

Step 6. (Output)

For j = 1 to |V|, print PERM(j).

End of The Enhanced Minimum Degree Algorithm.

D. IMPLEMENTATION

The implementation of the minimum degree algorithm uses the basic ideas of elimination graphs, reachable sets, indistinguishable vertices and is further enhanced through a

quotient graph partitioning of the initial elimination graph.

The graph G = (V,E) is stored using the adjacency array pair (XADJ,ADJNCY) in which the amount of storage used is O(|V|) and O(|E|), respectively. The amount of storage of the remaining arrays is O(|V|). ADJNCY gives the adjacent sets Adj(v) in contiguous storage locations for all v. XADJ(v) gives the location of the first element Adj(v) in ADJNCY. The resulting ordering array pair is PERM. The algorithm requires the determination of reachable sets for degree update.

Vertices identified as indistinguishable are merged together to form a supervertex. They are treated as one vertex as the algorithm continues. The current degrees of the vertices in the elimination graph are kept in the array DEG.

An initial partitioning of the graph is made to create a quotient graph which is a tree. Then, vertices of the elimination graph are selected only from within subsets of vertices making up a composite degree-1 vertex of the the quotient graph. In this way, fill-in can be guaranteed not to extend outside of the quotient graph. This technique helps keep fill-in low globally.

III. THE MINIMUM FILL-IN ALGORITHM AND VARIANTS

A. INTRODUCTION

In this chapter we introduce the minimum fill-in algorithm. In the minimum degree algorithm, the main concept is to select that row and column which yields the least total fill-in in the next active submatrix, while in the minimum fill-in algorithm the central idea is to select that row and column which introduces the least new fill-in.

Suppose $\{v_1, v_2, \ldots, v_{i-1}\}$ have been labelled. It is a simple matter, conceptually, to compute the new fill-in which would be introduced by selecting any vertex as the next vertex to be labelled as v_i. Fill(v_i) is the number of vertices v_j, $v_k \in$ Adj(v_i) such that v_j is not adjacent to v_k. In practice, this number can be computed easily if Adj(v) is kept as a linked list for all unlabelled vertices v. Below, the minimum fill-in algorithm is described followed by more detail on its implementation.

B. THE MINIMUM FILL-IN ALGORITHM

Algorithm : The Minimum Fill-in Algorithm.

Input : Elimination graph G(0) associated with a positive definite symmetric matrix.

Output: A permutation vector PERM( ) for the vertices of G(0)

Step 1. (Initialization)
Set i = 1.
Compute FILLIN(v) for all v ∈ V, the amount of new fill-in which would be created if v is eliminated.

Step 2. (Select minimum fill-in vertex)

In the elimination graph $G^{(i-1)} = (V_{i-1}, E_{i-1})$,

select a vertex $v_k \in V_{i-1}$ such that FILLIN(v)

is minimum.

Step 3. (Transform the graph)

Form the new elimination graph $G^{(i)} = (V_i, E_i)$

by eliminating the vertex v_k from $G^{(i-1)}$,

where all vertices adjacent to v_k in $G^{(i-1)}$

are connected to form a clique.

Set PERM(i) = k.

Step 4. (Update FILLIN array)

Update FILLIN(v) for all $v \in V_i$.

Step 5. (Loop or stop)

Set i = i+1.

If i > |V|, go to step 6.

Otherwise go to step 2.

Step 6. (Output)

For j = 1 to |V|, print PERM(j).

End of The Minimum Fill-in Algorithm.

C. IMPLEMENTATION OF THE MINIMUM FILL-IN ALGORITHM

The graph G = (V,E) is stored using the linked adjacency list arrays (XADJ,ADJNCY,LINK1) [Ref. 4] where the pointer XADJ(i) starts the adjacency list for vertex i, with ADJNCY containing a neighbor of vertex i and LINK containing the pointer to the location of the next neighbor of vertex i. The number of vertices in V is given by NEQNS. This data structure requires more space than the George and Liu algorithm; but, it requires space proportional to that necessary for carrying out one of the Gaussian elimination techniques. Thus, this is not really a limitation if storage is reused.

The purpose of the above algorithm is to find the minimum fill-in vertex for a general graph. It operates on the input graph as given by NEQNS and the linked adjacency list structure (XADJ,ADJNCY,LINK1), and returns the ordering in vector PERM.

The array FILLIN(v) stores the number of new fill-ins if vertex u is eliminated at the next step. FILLIN(v) is updated after each elimination step. This need be done only for those vertices which are a distance of two edges or less from u before the elimination step. The FILLIN value for vertices that have been eliminated is set negative.

Initial testing of the minimum fill-in algorithm has shown that it is much too time-consuming. Consequently, a restricted version has been developed which only searches among the minimum degree vertices for the minimum fill-in vertex. This is really a hybrid of the minimum degree algorithm and the minimum fill-in algorithm and is called the "min degree / min fill-in algorithm".

D. MIN DEGREE / MIN FILL-IN ALGORITHM

Algorithm : The Min Degree / Min Fill-in Algorithm.

Input : Elimination graph $G^{(0)}$ associated with a positive definite symmetric matrix.

Output: A permutation vector PERM( ) for the vertices of $G^{(0)}$

Step 1. (Initialization)
Set i = 1 and md = 0.

Step 2. (Lazy update of fill-in array FILLIN)
Find minimum degree mindeg.
If md ≠ mindeg, set md = mindeg and compute FILLIN(v) for all vertices v with deg(v) = md.
Otherwise go to step 3.

Step 3. (Selection)
In the elimination graph $G^{(i-1)} = (V_{i-1}, E_{i-1})$, select a vertex $v_k \in V_{i-1}$ which is of minimum fill-in among the minimum degree vertices.

Step 4. (Transform the graph)
Form the new elimination graph $G^{(i)} = (V_i, E_i)$ by eliminating the vertex v_k from $G^{(i-1)}$, where all vertices adjacent to v_k in $G^{(i-1)}$ are connected to form a clique.
Set PERM(i) = k.

Step 5. (Loop or stop)
Set i = i+1.

If i > |V|, go to step 6.
Otherwise go to step 2.

Step 6. (Output)
For j = 1 to |V|, print PERM(j).

End of The Min Degree / Min Fill-in Algorithm.

E. OTHER VARIANTS

The linked adjacency list structure used in the min degree / min fill-in algorithm makes it easy to try other ordering methods. Three additional methods will be tested : a simple minimum-degree algorithm, an initial minimum degree algorithm, and an initial ordering algorithm.

The "simple minimum-degree algorithm" is essentially the George and Liu algorithm but with a different data structure and without the reachable set and quotient graph enhancements. The "initial minimum degree algorithm" computes the degrees of the vertices in $G^{(0)}$ and selects vertices in increasing order of these degrees. No update of the initial degrees is ever made. The "initial ordering algorithm" just takes the vertices of $G^{(0)}$ in the order in which they are given, i.e., in the order in which the rows of the A matrix are given. This is a rather trivial algorithm but the time to create the compact data structure for sparse Gaussian elimination, included for the other algorithms is shown.

IV. COMPUTATIONAL EXPERIENCE AND DIFFICULTIES

To solve the sixteen test problems, we have used six methods to see which method is most effective. The methods are:

Method 1. The minimum degree algorithm of George and Liu.
Method 2. The minimum fill-in algorithm.
Method 3. The min degree / min fill-in algorithm.
Method 4. The simple minimum-degree algorithm.
Method 5. The initial minimum degree algorithm.
Method 6. Initial ordering algorithm.

Results for method 2 are not included because initial run times were excessive even for the smallest problems. As shown in Table III, two problems (JCAP, PAD) were not solved due to lack of computer memory. Time is CPU seconds necessary to obtain the ordering and to then create the compressed data structure used by George and Liu.

The minimum degree ordering algorithm of George and Liu is best. The simple minimum degree algorithm with linked data structures is almost as good - it is sometimes faster, and it is sometimes slower. But, new fill-ins created are always more than that of the minimum degree algorithm of George and Liu. The quotient graph enhancement of the George and Liu algorithm is probably responsible for its superior performance. Although storage requirements using the linked list data structure are greater, this only seemed to be a problem when the fill-in became very large. This structure gives a very flexible framework for trying different heuristics.

TABLE II

RESULTS OF ORDERING ALGORITHMS

Problem	Setup Time	Method 1 Fill	Method 1 Time	Method 3 Fill	Method 3 Time	Method 4 Fill	Method 4 Time	Method 5 Fill	Method 5 Time	Method 6 Fill	Method 6 Time
AIR	2.3	1.3	13.1	2.6	3.6	2.6	1.8	1.3	1.2	77.8	0.2
BUS	0.5	20.6	0.1	23.7	0.4	23.1	0.2	21.6	0.2	23.8	0.0
CCAL	3.2	8.4	5.7	11.9	11.5	11.9	2.5	8.4	2.4	11.9	0.1
CUPS	1.7	1.6	0.8	4.3	1.4	3.9	0.2	2.7	0.2	7.5	0.0
ELEC	21.4	1.6	1.1	30.9	16.8	30.5	1.9	29.8	1.7	13.0	0.1
FCAM	41.3	0.2	21.6	4.2	27.0	-	-	0.4	6.3	0.4	0.1
JCAP	-	-	-	-	-	-	-	-	-	-	-
NETTING	0.1	0.5	0.0	0.7	0.1	0.7	0.0	0.8	0.0	37.9	0.0
PASTOR1	27.7	4.1	0.6	35.6	8.1	35.4	0.9	30.7	1.3	25.8	0.1
PAD	-	-	-	-	-	-	-	-	-	-	-
PIES	28.0	9.1	2.2	-	-	-	-	30.1	3.0	68.0	0.2
PILOT	29.1	9.4	12.5	-	-	21.8	6.4	-	-	41.8	0.5
STEEL	21.4	5.6	7.3	-	-	12.5	8.2	22.1	16.4	22.1	0.4
STEINER1	0.2	56.4	0.4	59.1	3.5	61.8	0.9	64.0	1.1	56.2	0.0
STEINER2	1.3	69.3	2.3	-	-	-	-	-	-	70.7	0.2
TIGER	1.5	9.9	0.6	12.9	1.5	12.9	0.6	15.3	0.6	20.6	0.0

- means not solvable with 2 megabytes of memory.
"Fill" means new fill-in.

TABLE III

FILL-IN RESULTS

Problem	Initial Fill-in	Final Fill-in No Order	Final Fill-in Best Order	New Fill-in No Order	New Fill-in Best Order
AIR	20.8	98.6	22.1	77.8	1.3
BUS	55.7	79.5	76.4	23.8	20.6
COAL	26.1	38.0	34.5	11.9	8.4
CUPS	1.3	8.8	2.9	7.5	1.6
ELEC	2.0	15.0	3.6	13.0	1.6
FOAM	1.8	2.2	2.0	0.4	0.2
JCAP	-	-	-	-	-
NETTING	3.8	41.8	4.3	37.9	0.5
PASTOR1	1.7	27.4	5.7	25.8	4.1
PAD	-	-	-	-	-
PIES	1.9	69.8	11.0	68.0	9.1
PILOT	2.5	44.3	11.9	41.8	9.4
STEEL	3.0	25.1	8.7	22.1	5.6
STEINER1	31.0	87.2	87.4	56.2	56.4
STEINER2	19.1	89.8	89.4	70.7	69.3
TIGER	14.7	35.3	24.6	20.6	9.9

The min degree / min fill-in algorithm performed poorly. It is always slower, and the new fill-ins created are always more than that of the George and Liu algorithm. It cannot be recommended in any case and it is unclear why it performed so poorly.

Final fill-in for STEINER1 and STEINER2 in Table III is high, but these are artificial combinatoric problems. BUS is a small real-world problem with a best final fill-in of 76.4% which is much higher than the others. The probable reason for this is that BUS is a set partitioning problem.

TIGER with 24.6% final fill-in is also a set partitioning problem. This indicates that the projective algorithm might not work very well on many set partitioning and set covering problems since these seem to have a lot of intersections along rows. The lowest final fill-in found was 2.0% on FOAM. Six of the problems had less than 10.0% final fill-in. An interesting comparison can be made between the initial fill-in and the original densities of these problems. The initial fill-in can be compared to the density of matrix A which is about the same as the density for a basis in the simplex algorithm. This gives a idea of how hard creating L is compared to creating B^{-1} for the simplex algorithm. The initial fill-in of BUS, STEINER1 and STEINER2 was greater than 19.1% and they had densities greater than 6.7%. But, AIR, COAL and TIGER had more than 14.7% initial fill-in, while these had densities of only 1.2%, 1.2% and 4.1% respectively. The simplex algorithm might have a relatively greater advantage in these last cases.

V. CONCLUSIONS AND RECOMMENDATIONS

This thesis examined various heuristic algorithms for ordering symmetric linear systems of equations to minimize fill-in. The algorithms were tested on problems derived from linear programming problems in the manner of the projective algorithm for linear programming.

The minimum degree algorithm of George and Liu is the most effective of six tested methods. Solution times for this algorithm are sometimes better than and sometimes worse than the times using a simple minimum-degree algorithm, but new fill-ins created are always less. As a result of storage limitations, the simple minimum degree algorithm failed to solve two problems which the George and Liu algorithm solved. This should not be a problem in practice if storage is reused.

The minimum fill-in algorithm and the min degree / min fill-in algorithm used here do not perform as well as might be hoped. The minimum fill-in algorithm is too slow to use in most cases. Both algorithms involve more work than the minimum degree algorithm of George and Liu and the ordering produced is rarely much better than the one produced by the minimum degree algorithm of George and Liu.

The quotient graph enhancement of the George and Liu algorithm is probably responsible for its superior performance. Thus, future research should examine the use of the quotient graph and the min degree / min fill-in ideas since it may still be possible to significantly improve upon the new fill-ins obtained without sacrificing much computational speed. Different heuristics for creating quotient graphs should be examined, too.

LIST OF REFERENCES

1. Narendra K. Karmarkar, Technical Memorandum, 11216-840126-04, AT&T Bell Laboratories, 1984.

2. Gerald G. Brown, Richard D. Mcbride and R. Kevin Wood, Extracting Embedded Generalized Networks from Linear Progamming Problems, Mathematical Programming, Vol. 32, pp.11-31, 1985.

3. Sergio Pissanetsky, Sparse Matrix Technology, Academic Press, 1984.

4. Alan George and Joseph W-H Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Inc., 1981.

5. Curtis F. Gerald and Patrick O. Wheatley, Applied Numerical Analysis, Addison - Wesley Publishing Co., 1983.

6. Howard Anton, Elementary Linear Algebra, John Wiley & Sons, 1981.

7. Michael T. Heath, Numerical Methods for Large Sparse Linear Least Squares Problems, SIAM J. Sci. Comput., VOL. 5, NO. 5, pp. 497-513, September 1984.

8. Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, 1974.

9. Lawrence J. Corwin and Robert H. Szczarba, Multivariable Calcnus, Marcel Dekker, Inc. New York and Basel, 1982.

10. David G. Luenberger, Linear and Nonlinear Programming, Addison-Wesley Publishing Company, 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Department Chairman, Code 55Ws Department of Operations Research Naval Postgraduate School Monterey, California 93943-5100	1
4.	Professor Kevin Wood, Code 55Wd Department of Operations Research Naval Postgraduate School Monterey, California 93943-5100	5
5.	Professor Gerald G. Brown, Code 55Bw Department of Operations Research Naval Postgraduate School Monterey, California 93943-5100	1
6.	Professor Gordon H. Bradley, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
7.	Professor Richard Rosenthal, Code 55Ri Department of Operations Research Naval Postgraduate School Monterey, California 93943-5100	1
8.	Professor Arthur Schoentadt, Code 53Zh Department of Mathematics Naval Postgraduate School Monterey, California 93943-5100	1
9.	Professor Glenn Graves Graduate School of Management University of California, Los Angeles Los Angeles, California 90024	1
10.	Major Young Ho Seo 449 Seok-Nam Dong, Buk Gu, In Cheon, Seoul, Korea	15

END

FILMED

12-85

DTIC